#include <iostream>

using namespace std;

class node

{

public:

int data;

node \*left,\*right;

};

class tree

{

public:

node \*root,\*temp;

int height1(node \*T);

node \* mirror1(node \*T);

tree()

{

root=NULL;

}

void create();

void insert(node \*,node \*);

int height()

{

return(height1(root));

}

void inorder(node \*);

void min(node \*);

void search(node \*,int);

};

int tree::height1(node \*T)

{

if(T==NULL)

return(0);

if(T->left==NULL && T->right==NULL)

return(0);

return(max(height1(T->left),height1(T->right))+1);

}

void tree::create()

{

root=NULL;

char ch;

do{

temp=new node;

cout<<" enter data"<<endl;

cin>>temp->data;

temp->left=NULL;

temp->right=NULL;

if(root==NULL)

root=temp;

else

{

insert(root,temp);

}

cout<<"do u want to continue"<<endl;

cin>>ch;

}

while(ch=='y');

}

void tree::insert(node \*root,node \*temp)

{char ch1;

if(temp->data<root->data)

{

if(root->left==NULL)

root->left=temp;

else

insert(root->left,temp);

}

else if(temp->data>root->data)

{if(root->right==NULL)

root->right=temp;

else

insert(root->right,temp);

}

}

node \* tree::mirror1(node \*T)

{

node \*temp;

if(T==NULL)

return NULL;

else

{

temp=T->left;

T->left=mirror1(T->right);

T->right=mirror1(temp);

return T;

}

}

void tree::inorder(node \*root)

{

if(root!=NULL)

{

cout<<" "<<root->data;

inorder(root->left);

inorder(root->right);

}

}

void tree::min(node \*root)

{

while(root->left!=NULL)

root=root->left;

cout<<root->data;

}

void tree::search(node \* root,int x)

{

int flag=0;

while(root!=NULL)

{

if(x<root->data)

{

root=root->left;

}

else if(x>root->data)

{

root=root->right;

}

else if(x==root->data)

{

flag=1;

break;

}

}

if(flag==1)

cout<<"data found";

else

cout<<"not found";

}

int main()

{

tree t1;

int xx,op,x,c;

do

{

cout<<"\n\n1)Create\n2)Mirror";

cout<<"\n3)No of nodes in longest Path\n4)inorder display\n 5.minimum value\n6.Search\n7.Exit";

cout <<"\nEnter Your Choice :"<<endl;

cin>>op;

switch(op)

{

case 1:

t1.create();

break;

case 2:

cout<<"\n Original tree in inorder :\n";

t1.inorder(t1.root);

t1.root=t1.mirror1(t1.root);

cout<<"\n Mirrored tree in inorder :\n";

t1.inorder(t1.root);

cout<<"\nOriginal treerestored";

t1.root=t1.mirror1(t1.root);

break;

case 3:

cout<<"\nHeight = "<<t1.height();

break;

case 4:

t1.inorder(t1.root);

break;

case 5:

t1.min(t1.root);

break;

case 6:

cout<<"enter element to search";

cin>>x;

t1.search(t1.root,x);

break;

case 7:

exit(0);

}

}

while(op!=7);

return 0;

}